Agenda

- Fundamentals
  - Map and View
  - Basemap, Ground and Operational Layers
  - Layers
  - LayerViews
  - Widgets and UI
  - ArcGIS Platform
- Programming patterns
  - Interactivity with Input Manager
  - Working with Accessor
  - Promises
  - Loadable
Fundamentals
Map and View

Getting Started in 2D

• need a Map with data
• and a MapView with a container

```javascript
const map = new Map({
  basemap: "topo"
});

const view = new MapView({
  map: map,
  container: "viewDiv"
});
```
Map and View

Getting Started in 3D

- need a **Map** with data
- and a **SceneView** with a container

```javascript
const map = new Map({
    basemap: "topo"
});

const view = new SceneView({
    map: map,
    container: "viewDiv"
});
```
Map and View

- **Map** has information about the layers.
  - *What the world is composed of*
- **MapView** and **SceneView** displays each layer on the screen.
  - *A window on that world*
Map and View

View

- Renders and interacts with the model
- MapView
- SceneView

Model

- Describes the content of the map/scene
- Map
- Layers
Map and View

View
- Renders and interacts with the model

Model
- Describes the content of the map/scene

Diagram:
- View
  - MapView
    - LayerViews
  - SceneView
    - LayerViews
- Map
  - Layers
    - WebMap
    - WebScene
Map and View

View
Renders and interacts with the model

MapView
SceneView

LayerViews
LayerViews

Model
Describes the content of the map/scene

Map
Layers

WebMap
WebScene
Basemap, Ground, and Operational Layers

Layers are separated into 3 main groups.

- basemap
- ground
- operational layers

basemap and ground gives context to the operational layers.
Basemap, Ground, and Operational Layers

- `basemap` and `ground` can be set by well-known ids:

```javascript
const map = new Map({
 /*
   streets, satellite, hybrid, terrain, topo, gray,
   dark-gray, oceans, national-geographic, osm,
   dark-gray-vector, gray-vector, streets-vector, topo-vector,
   streets-night-vector, streets-relief-vector, streets-navigation-vector
  */
  basemap: "streets"

  /*
   world-elevation
  */
  ground: "world-elevation"
});
```
const map = new Map({
  basemap: {
    // Layers drawn at the bottom
    baseLayers: [
      new TileLayer({ url: baselayer })
    ],
    // Layers drawn on top
    referenceLayers: [
      new TileLayer({ url: refUrl })
    ],
  },
  ground: {
    layers: [
      new ElevationLayer({ url: elevationUrl })
    ]
  }
});
Basemap, Ground, and Operational Layers

- `basemap` can also be set by item id.
- Should probably be used in production.
Basemap, Ground, and Operational Layers

- `Map.layers` contains `Layer` objects with the operational data the user interacts with.

```javascript
const map = new Map({
  layers: [
    new MapImageLayer(...),
    new FeatureLayer(...)
  ]
});
```
Basemap, Ground, and Operational Layers

- a layer can only be in one place.
- there are layers in multiple places:
  - Pro: easy to swap a basemap with another
  - Pro: easy to reproduce a tree structure of the data
  - Pro: easy to manager a group of layer
  - Con: more places to look for changes
- They can be easily searched using `Map.allLayers`
  - contains the layers from every collection
  - I mean *everything*

```javascript
const layer = map.allLayers.find(layer => {
  return layer.title === "what I'm looking for";
});
```
Layers
TileLayer

- Layer that displays square images stitched together.
- It’s fast to display because the tiles are cached.
- It as a URL at points to a Map Service
const transportationLyr = new TileLayer({
  id: "streets",
  visible: false
});
WebTileLayer

- For use with machine serving map tiles
- Define the `level`, `column`, and `row` for map tiles
```javascript
const tiledLayer = new WebTileLayer({
  urlTemplate: "http://{subDomain}.tile.stamen.com/toner/{level}/{col}/{row}.png",
  subDomains: ["a", "b", "c", "d"],
  copyright: "Map tiles by <a href="http://stamen.com/">Stamen Design</a>, " +
    "under <a href="http://creativecommons.org/licenses/by/3.0">CC BY 3.0</a>. " +
    "Data by <a href="http://openstreetmap.org/">OpenStreetMap</a>, " +
    "under <a href="http://creativecommons.org/licenses/by-sa/3.0">CC BY SA</a>.",
});
```

**WebTileLayer**

**API sample**

**OSM**
GraphicsLayer

- Simplest layer to work with
- Symbolizes Graphic object on the view.
- a Graphic requires a geometry and a symbol.
- additionally a attributes and popup template.
const graphicsLayer = new GraphicsLayer({
    graphics: [graphic1, graphic2, graphic3]
});

// add a single graphic
graphicsLayer.add(graphic4);

// add an array of graphics
graphicsLayer.addMany([graphic5, graphic6, graphic7]);
const graphic = new Graphic({
  attributes: {
    id: 1,
    city: "Los Angeles"
  },
  geometry: { type: "point", x: xValue, y: yValue },
  symbol: {
    type: "simple-marker",
    style: 'circle',
    color: 'red',
    size: 10,
    outline: {
      color: "rgba(255, 255, 255, 0.5)"
    }
  },
  popupTemplate: {
    title: "My Awesome Graphic!",
    content: "{*}" // display all fields
  }
});
// add it to graphicsLayer
graphicsLayer.add(graphic);
FeatureLayer

- Displays features:
  - geometry
  - attributes
- Features are fetch from a service, or from a local collection
- Their geometry is the same for the entire layer.
- Cannot be symbolized individually
  - Feature.renderer
FeatureLayer

// Create via URL
const featureLayer = new FeatureLayer({
    url: "http://services6.arcgis.com/m3L8QUZ93HeaQzKv/arcgis/rest/services/BeerAndBurgerJoints/FeatureLayer"});

// Create via a Portal item
const featureLayer = new FeatureLayer({
    portalItem: {
        id: "b126510e440744169943fd8ccc9b0c4e"
    }
});
FeatureLayer - WebGL

- To display greater than 180,000 features
- We are already pushing limits of SVG
- Currently with hosted feature services
FeatureLayer - WebGL

```<script>
  var dojoConfig = {
    has: {
      "esri-featurelayer-webgl": 1
    }
  };
</script>
```

- That's it
MapImageLayer

- Displays layers and sublayers from Map Services
- Map Service can export map image given a bounding box
- Simplified API for dynamic layer infos
  - sublayers
const layer = new MapImageLayer(
  url: "https://sampleserver6.arcgisonline.com/arcgis/rest/services/USA/MapServer",
  sublayers: [
    { id: 0, visible: true },
    { id: 1, visible: true },
    { id: 2, visible: true, definitionExpression: "pop2000 > 1000000" },
    { id: 3, visible: false }
  ]
));
LayerViews
LayerViews

View
Renders and interacts with the model

MapView
SceneView

LayerViews

Model
Describes the content of the map/scene

Map
Layers

WebMap
WebScene
LayerViews

- **LayerViews** renders the layers on the screen.
- **LayerView** has limited API so far.
- Give info about layer rendering
- Give access to data displayed on the screen
  - Features
  - Elevation data
LayerViews

- access a layerview with `View.whenLayerView()`
- or `View.allLayerViews`
LayerViews

Run Pen
LayerViews

- FeatureLayer and LayerViews can be queried
  - `featureLayer.queryFeatures()` - query features on the service
  - `featureLayerView.queryFeatures()` - query features on the client
Widgets and UI
Widgets

• New design and user experience
  ▪ Accessibility
  ▪ UX
  ▪ Extensibility
• New architecture
  ▪ Views + ViewModels
Widgets Styling

- Styling
  - Sass
  - Styling Guide
  - CSS Themes
  - Github JSAPI Resources SASS
All The Widgets!

- Out of the Box Widgets Demo
Widgets - View Model

- Logic of the widget separated from the representation
- Provide APIs to support view
- ZoomViewModel
Widgets - Views

- Extends `esri/widgets/Widget` Widget
- Rely on ViewModel
- Focus on UI
- Views' source code available in the SDK
- View's can be rewritten in any framework using ViewModels.
- Simple Widget View Demo
Widgets - Development Doc

- Widget Development Doc
- Custom Widget Tutorial
UI

- Managed overlay to place widgets over the view.
- Padding
  - UI.padding defines spacing for widgets/UI.
  - View.padding defines spacing for center, and extent, etc. (work off a subsection of the full view)
- Well known widgets can be directly added or removed from the view
- UI Guide
- UI Padding Demo
UI

- API to add widgets or any DOM element to the 4 corners of the view

```javascript
const view = new MapView(
  //...
);

const legend = new Legend(
  //...
);

view.ui.add(legend, "top-left");
```

UI Manual Positioning
Popups

- Popups are responsive
- First entry point to detailed data

```javascript
// basic popup
const featureLayer = new FeatureLayer({
  outFields: ["*"],
  popupTemplate: {
    title: "Name: {STATE_NAME}",
    content: "{*}"
  }
});
```
Popups - Fields and Aliases

- First entry point to detailed data
- **Popup Content Doc**

```javascript
content: [
  {
    type: "fields",
    fieldInfos: [
      {
        fieldName: "POP2000",
        visible: true,
        label: "Population for year 2000",
        format: {
          places: 0,
          digitSeparator: true
        }
      },
      ...
    ]
  }
]"
Popups - Fields and Aliases

- Format dates

```json
{
  fieldName: "FAKEDATE",
  visible: true,
  label: "Fake Date Field",
  format: {
    dateFormat: "short-date"
  }
}
```
const featureLayer = new FeatureLayer({
    outFields: ["*"],
    popupTemplate: {
        title: "Name: {STATE_NAME}"
        content: `<section>
            <h4>{STATE_ABBR}</h4>
            <hr />
            <ul>
                <li>Year 2000 Pop: {POP2000}</li>
                <li>Year 2007 Pop: {POP2007}</li>
                <li>Total Households: {HOUSEHOLDS}</li>
            </ul>
        </section>`
    });
Popups - MediaInfos

• Charts

```json
{
  type: "media",
  mediaInfos: [
    {
      title: "<b>Population</b>",
      type: "column-chart",
      caption: "",
      value: {
        theme: "BlueDusk",
        fields: [ "POP2000", "POP2007" ]
      }
    }
  ]
}
```
// PopupTemplate
{
    title: '{Name}',
    content: '{*}',
    actions: [{
        id: 'alcohol-details',
        className: 'esri-icon-description',
        title: 'Events'
    }]
}
Popups - DockOptions

```javascript
view.popup.set("dockOptions", {
  breakpoint: false,
  buttonEnabled: false,
  position: "bottom-center"
});
```

- DockOptions Doc
- DockOptions demo
popupTemplate: new PopupTemplate({
  title: "Arcade Example",
  content: "\{expression/total\} total items.",
  expressionInfos: [{
    "name": "total",
    "title": "Total Items",
    "expression": "$feature.type1 + $feature.type2 + $feature.type3",
    "returnType": "number"
  }]
})

- Arcade Info
- Arcade Demo
Popups - Advanced Setting Content

```javascript
popupTemplate: new PopupTemplate({
    content: getChart,
    title: function (event) {
        var graphic = event.graphic;
        return graphic.attributes.title;
    },
})
```

Popup Advanced Demo
Popups - Setting via promises

```
view.popup.open({
  promises: [promise1, promise2, promise3],
  location: event.mapPoint
});
```

Popup Promises Demo
ArcGIS Platform
ArcGIS Platform

- redesigned API
  - Portal API
- access portal information: basemaps, featuring content
- query items, users, groups
- loading items like layers, webmap and webscene
- creating, deleting and updating items
ArcGIS Platform

- Loading a WebScene

```javascript
const scene = new WebScene({
  portalItem: {
    id: "082c4fd545104f159db39da11ea1e675"
  }
});

const view = new SceneView({
  map: scene,
  container: "viewDiv"
});
```
ArcGIS Platform

- Loading a layer from an item.

```javascript
const promise = Layer.fromPortalItem({
  portalItem: {
    id: '8444e275037549c1acab02d2626daaee',
    portal: {
      url: 'https://myorg.maps.argis.com'
    }
  }
})
.then(layer => {
  // Adds the layer to the map once it loads
  map.add(layer);
})
.otherwise(error => {
  // handle the error
});
```

- demo
const portal = new Portal();

// Setting authMode to immediate signs the user in once loaded
portal.authMode = 'immediate';

// Once loaded, user is signed in
portal.load()
  .then(() => {
    // Create query parameters for the portal search
    const queryParams = new PortalQueryParams({
      query: 'owner:' + portal.user.username,
      sortField: 'numViews',
      sortOrder: 'desc',
      num: 20
    });

    // Query the items based on the queryParams created from portal above
    portal.queryItems(queryParams).then(createGallery);
  });

• demo
Programming patterns
Interactivity with view events

- Use view events to interact with the view
- List of events
- You can stop the propagation of the event to prevent the default behavior

```javascript
view.on("drag", event => {
  // user won't be able to drag
  event.stopPropagation();
})
```

Drag Demo
Interactivity with view events

- Access the features on click

```javascript
view.on("click", ({ x, y }) => {
    const screenPoint = {x, y};
    view.hitTest(screenPoint)
        .then(response => {
            // do something with the result graphic
            const graphic = response.results[0].graphic;
        });
});
```

- API Sample
- Click Demo
**goTo() with View**

- Sets the view to a given target.
  - Navigate to a geometry/feature/location
- `view.goTo()` Demo
Collections

- esri/core/Collection
- Collection Doc
- Collection Demo
Working with Accessor

- Objects are have properties that can be:
  - read and set
  - or read-only
  - constructor arguments
  - watchable
Accessor - property access

```javascript
console.log(layer.opacity);
console.log(layer.title);

layer.opacity = 0.5;
layer.title = "My test layer";

// setting multiple values
layer.set({
    opacity: 0.5,
    title: "My test layer"
});

// accessing the value of a deep property
view.get("map.basemap.title");
view.set("map.basemap.title", "new title");
```
Accessor - property watching

```javascript
mapView.watch("scale", (newValue, oldValue, property, target) => {
    console.log(`scale changed: ${newValue}`);
});

mapView.watch("map.basemap.title", (newValue, oldValue, property, target) => {
    console.log(`new basemap title: ${newValue}`);
});

mapView.watch("ready, stationary", (newValue, oldValue, property, target) => {
    console.log(`property ${property}: ${newValue}`);
});

watchUtils.whenTrue(view, "stationary", () => {
    console.log("view is stationary");
})
```

**watchUtils**
Accessor - autocasting and single constructor

// 4.x
{
    type: "simple-marker",
    style: 'square',
    color: 'red',
    size: 10,
    outline: {
        color: 'rgba(255, 255, 255, 0.5)'
        width: 4
    }
}
});

// 3.x
new SimpleMarkerSymbol(SimpleMarkerSymbol.STYLE_SQUARE, 10,
    new SimpleLineSymbol(SimpleLineSymbol.STYLE_SOLID,
        new Color([255,0,0]), 4),
        new Color([255,255,255,0.25]));
Accessor - autocast with Collections

Demo Autocast Collection
Promises
Promises

- All asynchronous methods return a promise, no more events
- The basic pattern looks like this:

```javascript
layer.queryFeatures(query).then(handleResult).catch(handleError);
```
Promises with async/await

- work with native promises

```javascript
const doQuery = async (query) => {
  const results = await layer.queryFeatures(query);
  const transformedResults = results.map(transformData);
  return transformedResults;
}
```
Promises

- Classes may be Promise
  - Load resources
  - Asynchronously initialized `Layer`, `WebMap`, `WebScene`, `View`
  - `view.then()` replaces `map.on('load', ...)`
  - We add `when()` to the API.

```javascript
const map = new Map({ ... })

view = new SceneView({
  map: map,
  // ...
});

view.when(() => {
  // the view is ready to go
});
```
view.when(() => {
  return view.whenLayerView(map.findLayerById("awesomeLayer"));
})
  .then(layerView => {
    return watchUtils.whenFalseOnce(layerView, "updating");
  })
  .then(result => {
    const layerView = result.target;
    return layerView.queryFeatures();
  })
  .then(doSomethingWithFeatures)
  .catch(errorHandler);
Loadables

- brings better control, and scheduling of loading resources.
- extension of `esri/core/Promise`
- in 3.x, instanciating a layer loads it. in 4.0, it's an explicit call
- the views automatically loads the map and its layers
Loadables

- **WebMap / WebScene** need to load:
  - the portal item
  - the layer module
  - the layer’s item
- **MapView / SceneView** need to load:
  - the map
  - the layers
In a single page application, get a feature from a FeatureLayer from a WebMap without displaying it, ASAP!

```javascript
const webmap = new WebMap({
    portalItem: {
        id: 'affa021c51944b5694132b2d61fe1057'
    }
});
webmap.load().then(() => {
    return webmap.getLayer('myFeatureLayerId').load();
}).then(featureLayer => {
    return featureLayer.queryFeatures({
        where: 'OBJECTID = 1'
    });
}).then(result => {
    displayDetails(result.features[0]);
}).otherwise(error => {
    console.error(error);
});
```
Questions?

Matt Driscoll (mdriscoll@esri.com) (@driskull)
Rene Rubalcava (rrubalcava@esri.com) (@odoenet)